Symbolic Test generation using verification

Thierry Jéron
Irisa/Inria Rennes, France
http://www.irisa.fr/vertecs/

Adapted from

B. Jeannet, T. Jéron, V. Rusu, E. Zinovieva,
Symbolic Test Selection based on Approximate Analysis,
in TACAS'05, LNCS 3440, Edinburgh (Scottland), April 2005.

Vlad Rusu, Hervé Marchand, Thierry Jéron,
Automatic Verification and Conformance Testing for Validating Safety Properties of Reactive Systems,
in Formal Methods 2005 (FM05), July 2005.

Outline

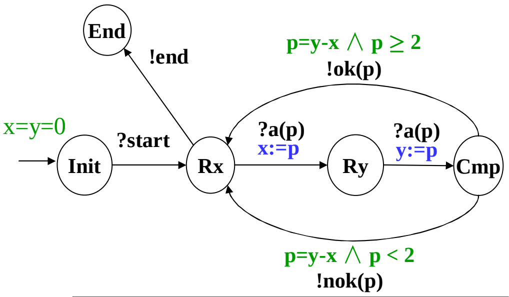
- **The *ioSTS* model**
- **Conformance Testing Theory with ioco**
- **Test selection using approximate analysis**
- **Conclusion**

2

1.The *ioSTS* model

S = (V_S, Θ_S, Σ, T_S) with

- V_S: vector of variables $\ni$ loc
  valuations v_i of v_i in Dom(v)
- Θ_S: initial condition
- $\Sigma = \Sigma_! \cup \Sigma_? \cup \Sigma_\tau$:
  alphabet of actions
  with comm. parameters p$\in$ P
  sig(a): type of comm. param
- T_S: transition relation
  [a(p) : G(v_S,p); v_S:=A(v_S,p)]
  action guard assignment
  also noted [a,p,G,A]



V_S = {loc,x,y} Dom(x)=Dom(y)= Z
Θ_S = {x=y=0}
$\Sigma_!$ = {!end,!ok,!nok} $\Sigma_?$ = {?start, ?a}
P = {p}

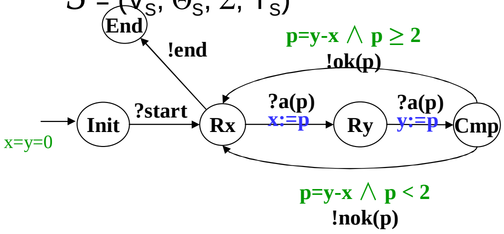
Hyp:

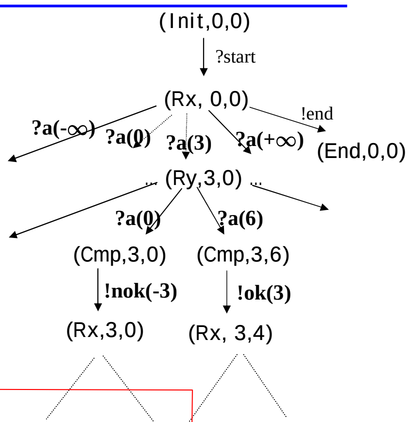
- satisfiability of guards is decidable.
- Θ_S has a unique solution in Dom(V_S)

3

ioLTS semantics of *ioSTS*

S = (V_S, Θ_S, Σ, T_S)





[[S]] = S = (Q,Q_0, Λ,$\rightarrow$) with

- Q = Dom(V_S) = $\times_{v \in V_S}$ Dom(v)
  i.e. q = <v_0, ...v_n> is a vector of valuations
- Q_0= {q_0} where q_0 =<v_0, ...v_n> is the unique solution to Θ_S
- Λ = {<a,π> | a $\in$ $\Sigma \wedge \pi \in$ Dom(sig(a))}
- $\rightarrow$ is defined by: [a,p,G,A] $\in$ T, q $\in$ Q, $\pi \in$ Dom(sig(a)), G(q,π)
  q - a(π)$\rightarrow$ q' = A(q,π)

4

Runs, traces

S = [[$\mathcal{S}$]] is an ioLTS thus

Runs($\mathcal{S}$) = Runs(S): $q_0 \rightarrow^{a_1(\pi_1)} q_1 \rightarrow^{a_2(\pi_2)} q_2 \ldots \in Q_0.(\Lambda.Q)^*$

represent executions from the initial state

Tr($\mathcal{S}$) = Tr (S) : $proj_{\Lambda_{vis}}$(runs(S))

projection of runs on visible actions

STr($\mathcal{S}$)= STr(S)= Tr(Δ(S))

5

Deterministic ioSTS

Def: an *ioSTS* $\mathcal{S}$ is *deterministic* iff its semantics is a deterministic ioLTS [[$\mathcal{S}$]]

Undecidable problem

Def: $\mathcal{S}$ is *syntactically deterministic* if $\mathcal{S}$ has no internal action and $\forall$ action $a \in \Sigma$, $\bigcap_{t=[a,\, p, G_t, A_t]} G_t = \emptyset$

Prop: $\mathcal{S}$ is syntactically deterministic $\Rightarrow$ [[$\mathcal{S}$]] is deterministic

NB: Improvement using over-approximate reachability analysis

Let $reach^{\alpha} \supseteq reach$, if $\forall$ action a, $\bigcap_{t=[a,\, p, G_t, At]} G_t \cap reach^{\alpha} = \emptyset$

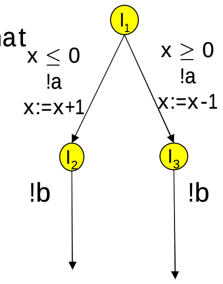
then $\mathcal{S}$ is deterministic

(e.g. $reach^{\alpha} \subseteq l_1 \Rightarrow x \neq 0$)

6

Determinisation of *ioSTS*

Problem: given an *ioSTS S*, construct an *ioSTS det(S)* such that *det(S)* is deterministic and Tr(*det(S)*)) = Tr(*S*)



Determinisation of *ioSTS* into *ioSTS* is not always possible (deterministic *ioSTS* are a proper subclass of *ioSTS*)

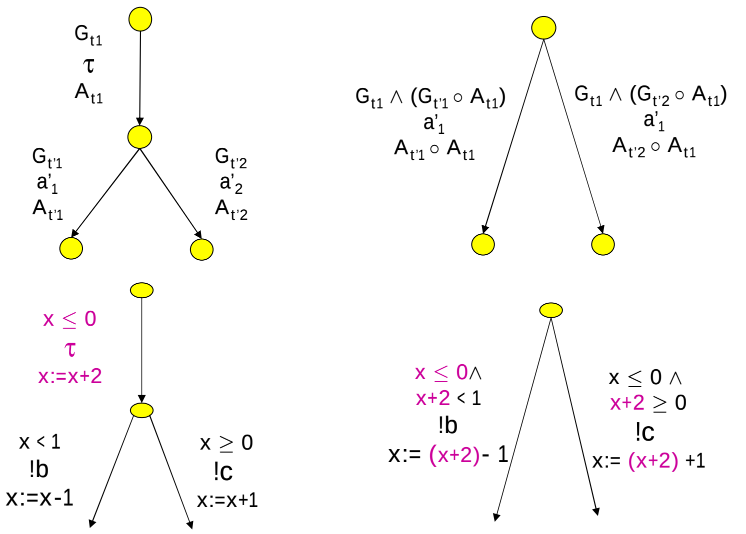
→ Syntactical heuristics with restrictions needed for termination:

- No internal loop for « ε-closure »
- Finite lookahead for « subset-construction »

7

Symbolic ε-closure

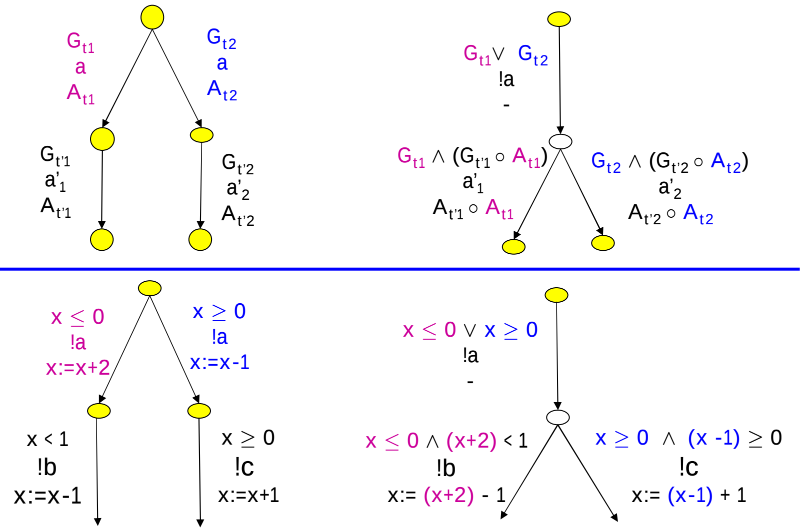
Idea: propagate assignments to next observable actions



8

Determinisation heuristic

Idea: propagate assignments on next transitions until decision



9

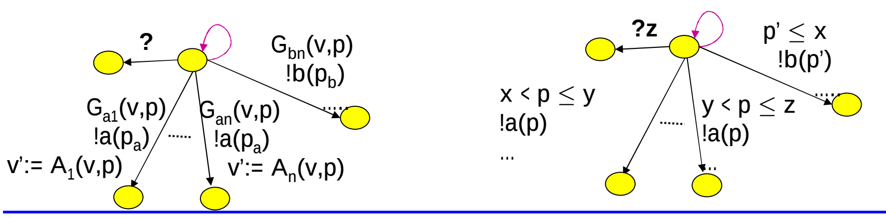
Quiescence

Problem: explicit quiescence by adding loops with $!\delta$
in all quiescent states (no output is feasible)

Transform S syntactically in $\Delta(S)$ such that $[[\Delta(S)]] = \Delta([[S]])$
Augment guard model with universal quantification
not a real problem

$\bigwedge_{a \in \Sigma_! \cup \Sigma_\tau} \neg (\bigvee_{t=[a,p,G,A]} \exists \pi, G(v,\pi))$
$!\delta$

$\neg (\exists \pi, x < \pi \leq y \vee y < \pi \leq z) \wedge$
$\neg (\exists \pi, \pi \leq x)$
$!\delta$



10

Simplyfying asumptions

- *ioSTS* are supposed to be deterministic
- Quiescence is not considered

2. Conformance Testing Theory with ioco [Tretmans 96]

- Specification: known ioLTS S (semantics of an ioSTS)
- Implementation: unknown ioLTS I
- Conformance: I ioco S
- Test cases : ioSTS TC + Verdict variables
  - Execution: parallel composition $\Delta(I) \parallel TC$
  - Verdicts: TC fails I
- Test generation: gen_test: S $\rightarrow$ TS= {TC_1 TC2,...}

  Requested Properties of TS: TS fails I $\leftrightarrow \neg$ I ioco S

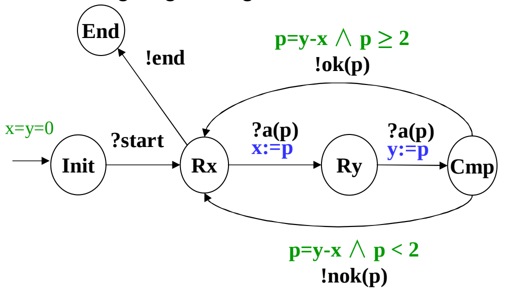
  (soundness, limit exhaustiveness)

Simplification: an automata/language point of view

Conformance relation

I ioco S , $\forall \sigma \in STr(S)$, $Out(\Delta(I)$ after $\sigma) \subseteq Out(\Delta(S)$ after $\sigma)$

$S = (V_S, \Theta_S, \Sigma, T_S)$



Conformant traces, e.g.
?start . ?a(4) . ?a(6) . !ok(2)
?start . !end
?start. ?start
(unspecified input allowed)

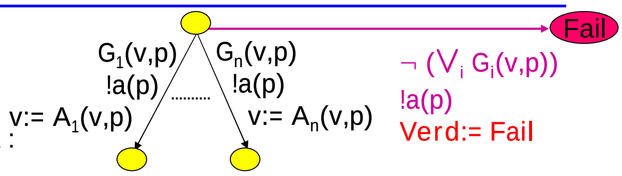
Non-conformant traces, e.g.
?start . ?a(5) . ?a(7) . !ok(3)
?start . ?a(6) . ?a(8) . !nok(2)

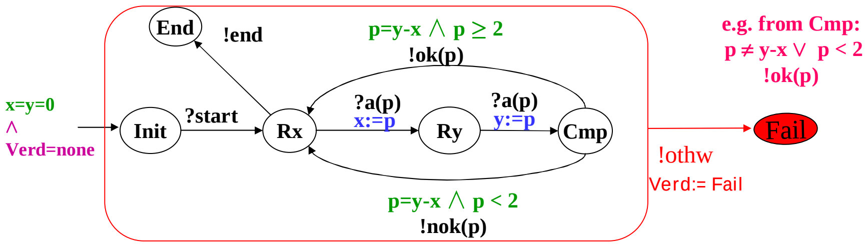
Prop: I ioco S $\Leftrightarrow STr(I) \cap [STr(S) . \Lambda_!^\delta \setminus STr(S)] = \emptyset$
non-conformant behaviours

13

Canonical Tester Can(S) : observer of non-conformant behaviours

1. Add new variable Verd with initial value none i.e. $\Theta \wedge$ Verd = none
2. $\forall$ ouput !a, $\forall$ t carrying !a :





$STr_{Fail}(Can(S)) = STr(S). \Lambda_!^\delta \setminus STr(S)$

$\Rightarrow$ I ioco S $\Leftrightarrow STr(I) \cap STr_{Fail}(Can(S)) = \emptyset$

14

Test cases, test execution, verdicts and properties

Test Case: deterministic ioSTS TC= $(V_{TC}, \Theta_{TC}, \Sigma, T_{TC})$

+ verdict variables Verd ∈ {none, Fail, Pass, Inconc, ...}

plays the role of an observer delivering verdicts

$Tr_{Fail}(TC)$: $\{\sigma \in Tr(TC) \mid TC \text{ after } \sigma \in Fail\}$

Test suite: (infinite) set of test cases TS= $\{TC_1, TC_2, \ldots\}$

Test execution: TC || Δ(I) synchronization on common actions

Possible rejection of I by TC:

TC fails I , $STr(I) \cap Tr_{Fail}(TC) \neq \emptyset$

Test suite properties

Possible rejection by a TC should correspond to non-conformance and vice-versa

TC fails I ⇔ $STr(I) \cap Tr_{Fail}(TC) \neq \emptyset$

I ioco S ⇔ $STr(I) \cap Tr_{Fail}(Can(S)) = \emptyset$

TS is sound , $\forall I, (I \text{ ioco } S \Rightarrow \forall TC \in TS, \neg TC \text{ fails } I)$

⇔ $\bigcup_{TC \in TS} Tr_{Fail}(TC) \subseteq Tr_{Fail}(Can(S))$

TS is exhaustive , $\forall I, (\forall TC \in TS, \neg TC \text{ fails } I \Rightarrow I \text{ ioco } S)$

⇔ $\bigcup_{TC \in TS} Tr_{Fail}(TC) \supseteq Tr_{Fail}(Can(S))$

3. Test selection for *ioSTS*

TS = {Can(S)} is a sound and exhaustive test suite but

- has too many (infinite) behaviours
- does not allow to control the implementation during testing

⇒ Test selection

- renounce to exhaustiveness in practice,
  select a finite TS likely to discover non-conformances
- focus on targetted behaviours of Can(S)
- use test purposes

17

Test purposes

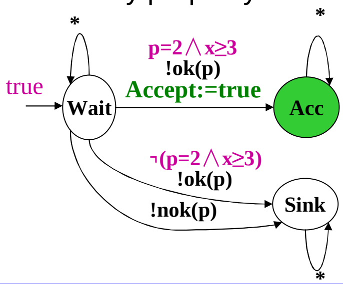
$TP = (V_S \cup V_{TP}, \Theta_{TP}, \Sigma, T_{TP})$

Observer of actions and variables of S

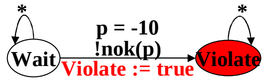
$[a(p) : G(v_S, v_{TP}, p); v_{TP} := A(v_S, v_{TP}, p)] \in T_{TP}$

Hyp : complete and deterministic

TP+: reachability property

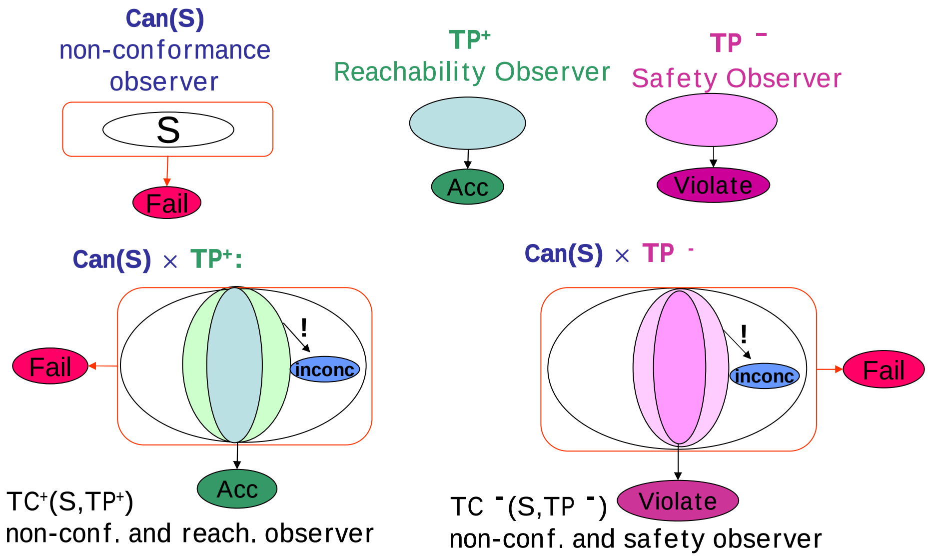


TP -: negation of safety property



18

General principle



Syntactical product $Can(S) \times TP$

$Can(S) = (V_{Can(S)}, \Theta_{Can(S)}, \Sigma, T_{Can(S)})$

$G_1(v_S,p)$
$a(p)$
$v'_S := A_1(v_S,p)$

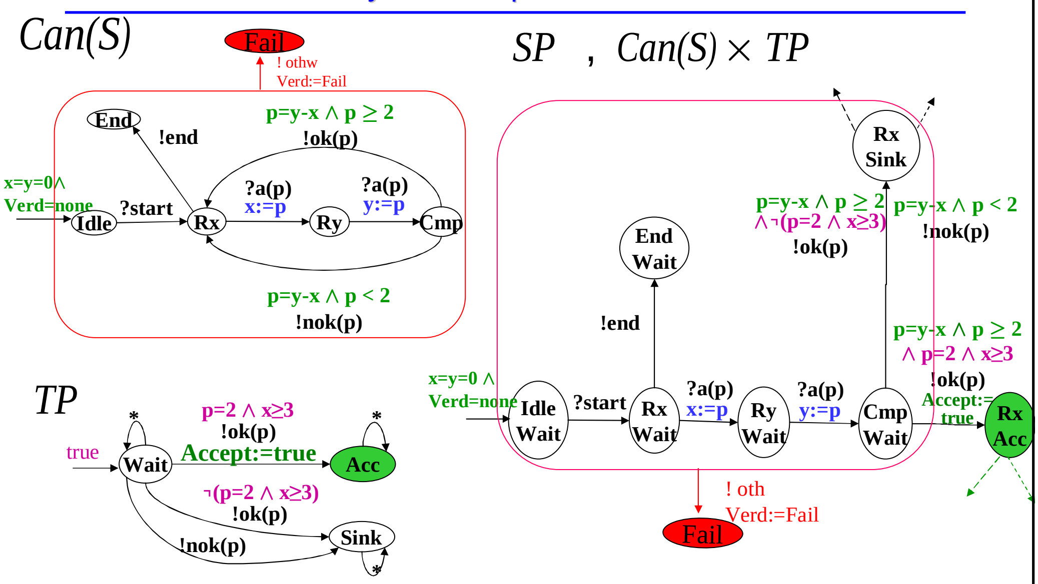
$TP = (V_S \cup V_{TP}, \Theta_{TP}, \Sigma, T_{TP})$

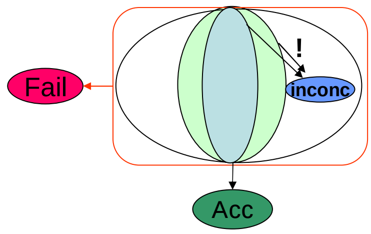
$G_2(v_S,v_{TP},p)$
$a(p)$
$v'_{TP} := A_2(v_S, v_{TP},p)$

$$SP = Can(S) \times TP = (V_{Can(S)} \cup V_{TP}, \Theta_{Can(S)} \cap \Theta_{TP}, \Sigma, T_{Can(S)\times TP})$$

$G_1(v_S,p) \wedge G_2(v_S, v_{TP}, p)$
$a(p)$
$\langle v'_S; v'_{TP}\rangle :=$
$\langle A_1(v_S,p);A_2(v_S, v_{TP},p)\rangle$

Syntactical product





After product we get

- Tr(SP) = Tr(Can(S))
- $Tr_{Fail}(SP) = Tr_{Fail}(Can(S))$
- $Tr_{Accept}(SP) = Tr_{Accept}(TP) \cap STr(S)$

SP is both a non-conformance and reachability observer
but has too much behaviours: (Tr(Can(S))

Goal of selection:

focus on $Tr_{Accept}(TP) \cap STr(S)$,
detect unfeasible traces to Accept
Amounts to compute co-reach(Accept)
Undecidable ⇒ over-approximate

Syntactical Test Selection (1)

1. Assignment of Pass verdicts

Pass: $Tr_{Accept}(SP)$ → Observer of $Tr_{Accept}(SP)$

$Can(S) \times TP$

G(v,p)
a(p)
v:= A(v,p)

⇒

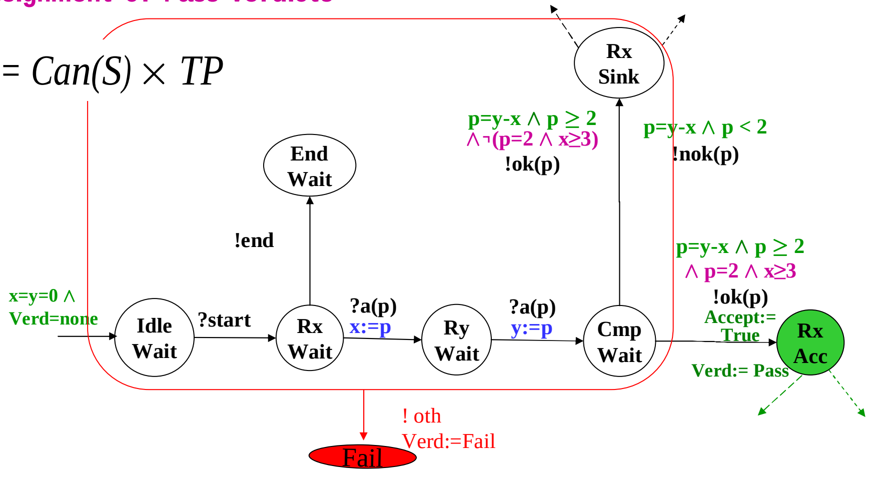
G(v,p) ∧ Verd = none
a(p)
v:= A(v,p)
Verd :=if A_{Accept} then Pass
else Verd

CTG

Syntactical Test Selection (1)

1. Assignment of Pass verdicts

$SP = Can(S) \times TP$



Syntactical Test Selection (2)

2. Selection and assignment of Inconc verdicts

coreach(Accept) not computable ⇒ compute over-approximation:

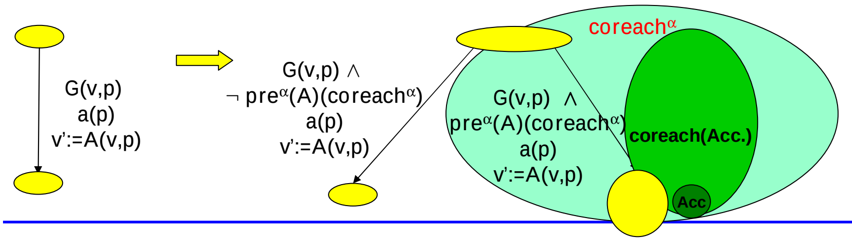
$coreach^{\alpha} \supseteq coreach(Accept)$

$\forall$ assignment A, $pre^{\alpha}(A)(coreach^{\alpha}) \supseteq pre(A)(coreach^{\alpha})$

Idea: $pre^{\alpha}(A)(coreach^{\alpha})$ = Nec. Cond. to go into $coreach^{\alpha}$

$\neg pre^{\alpha}(A)(coreach^{\alpha})$ = Suf. Cond. to go outside $coreach^{\alpha}$

$\subseteq$ outisde coreach(Accept)

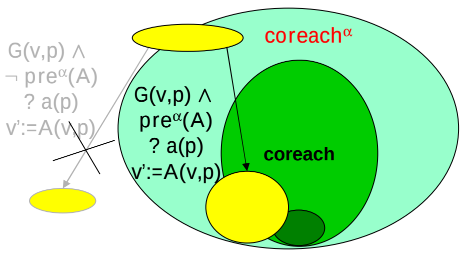


Syntactical test selection (3): guard strengthening

Rule for inputs of *S*:

keep conditions leading to $coreach^{\alpha}$,

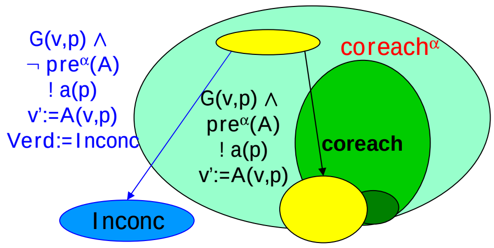
cut other ones (controllable):



Rule for outputs of *S*

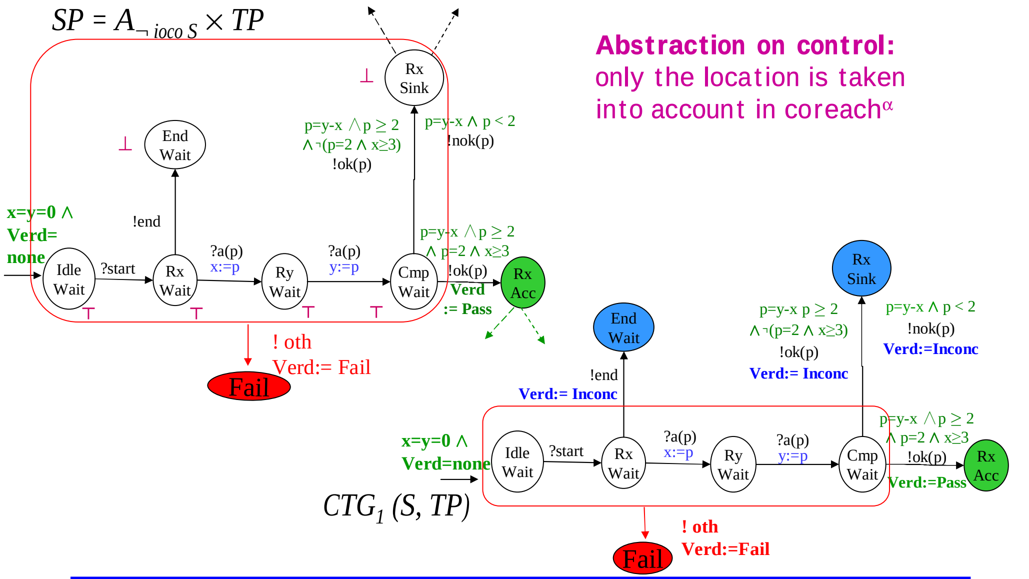
keep all conditions (uncontrollable),

those leading outside $coreach^{\alpha}$ produce Inconc:

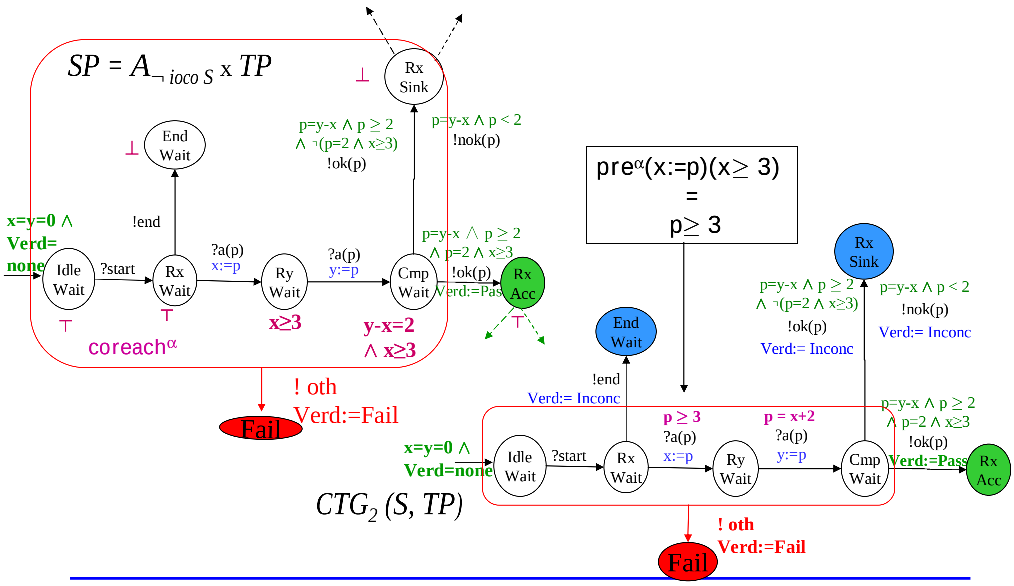


Test selection: example
1^{st} over-approximation : control

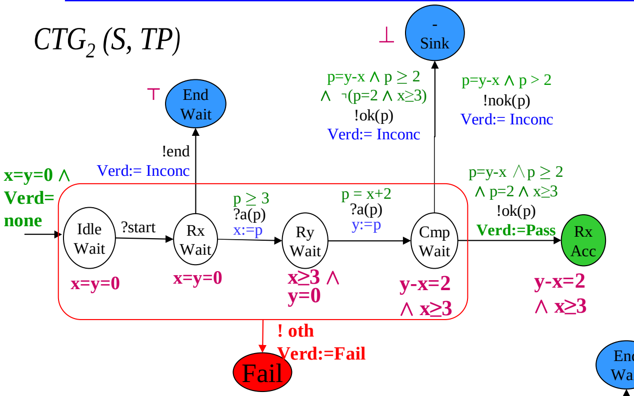
Abstraction on control:
only the location is taken into account in coreach$^{\alpha}$



Test selection: example
2^{nd} approximation computed by NBAC (convex polyhedra)

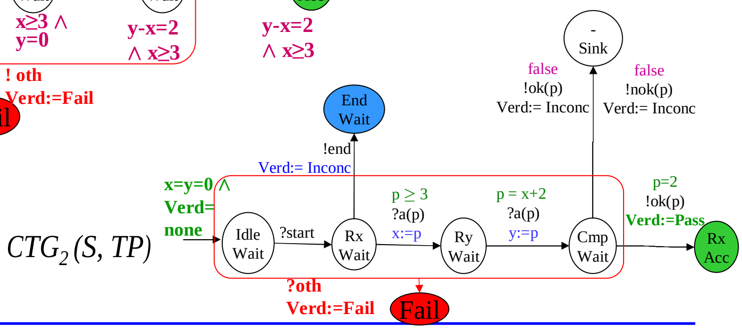


Simplification: over-approximation reach$^\alpha$ of reach(Θ)



Simplify guards
according to reach$^\alpha$
(false ⇒ cut)

NB: semantics is unchanged

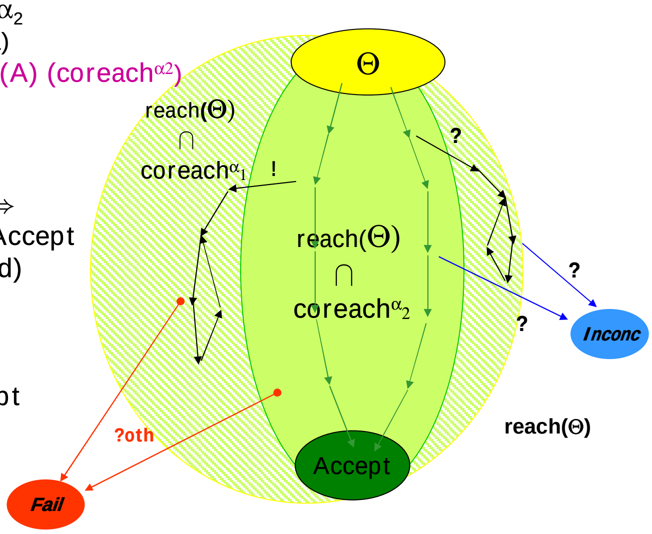


Consequences of over-approximation on test cases

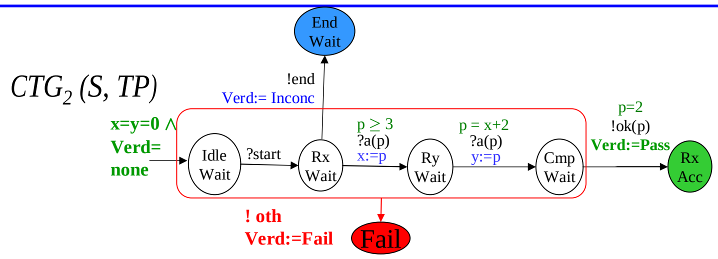
For two abstractions α_1 and α_2
(e.g. α_1: control vs α_2: polyhedra)
pre$^{\alpha 1}$ (A) (coreach$^{\alpha 1}$) ⊇ pre$^{\alpha 2}$ (A) (coreach$^{\alpha 2}$)
⇒
Tr(CTG_1) ⊇ Tr(CTG_2)

Less precise approximation ⇒
- More infeasible traces to Accept
- More fail verdicts (all sound)

Limit cases:
- exact analysis:
  best guiding to Accept
- no analysis:
  no guiding to Accept



Test execution



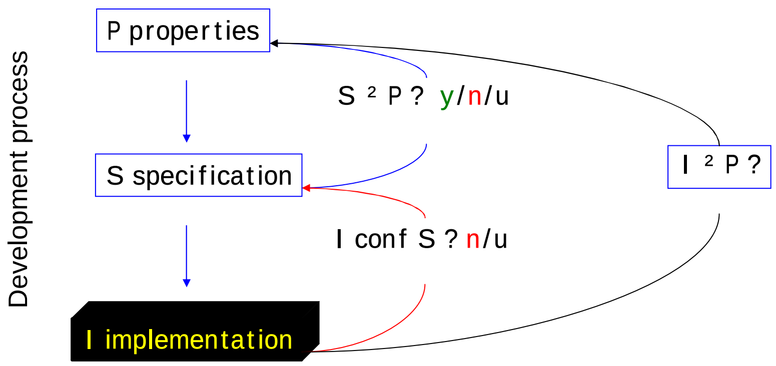
Inputs: [? a(p) : G(v,p); v:=A(v,p)] : v is known,
choose π s.t. G(v,π),by constraint solving
send a(π), assign v:=A(v, π)

Outputs: [! a(p) : G(v,p); v:=A(v,p)]: v is known,
receive a(π)
evaluate G(v,π)
if true, assign v:= A(v,π) (input complete)

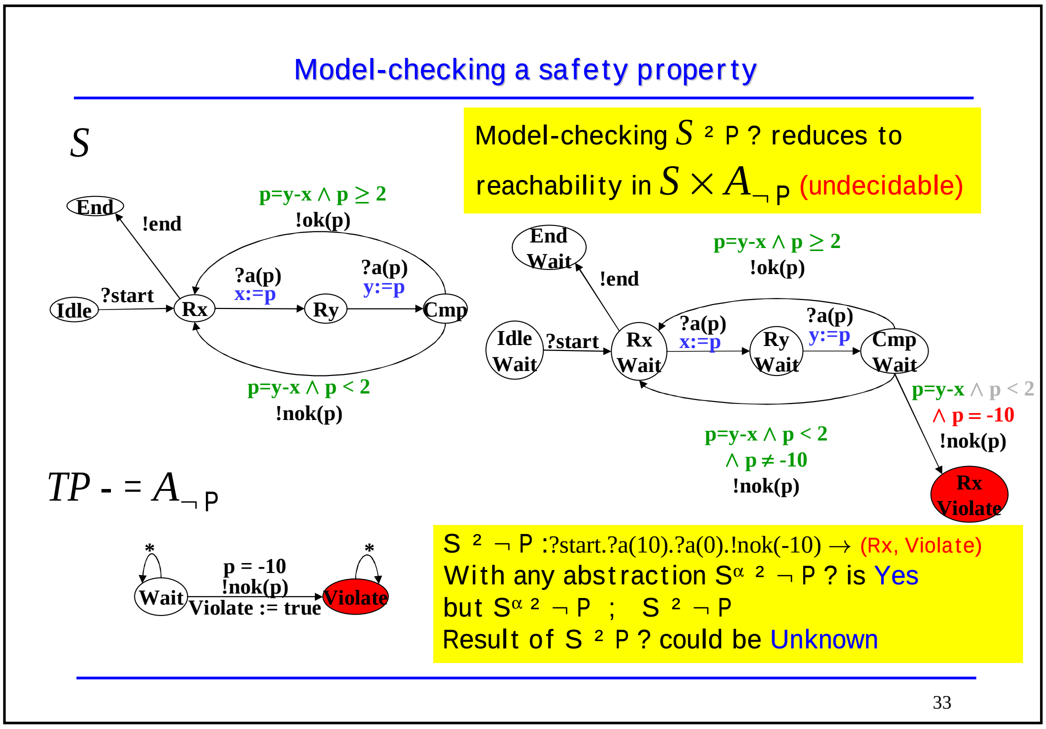
?start . ?a(4) . ?a(6) . !ok(2) : Pass
?start . !end : Inconc
?start . ?a(5) . ?a(7) . !ok(3) : Fail
?start . ?a(6) . ?a(8) . !nok(2) : Fail

31

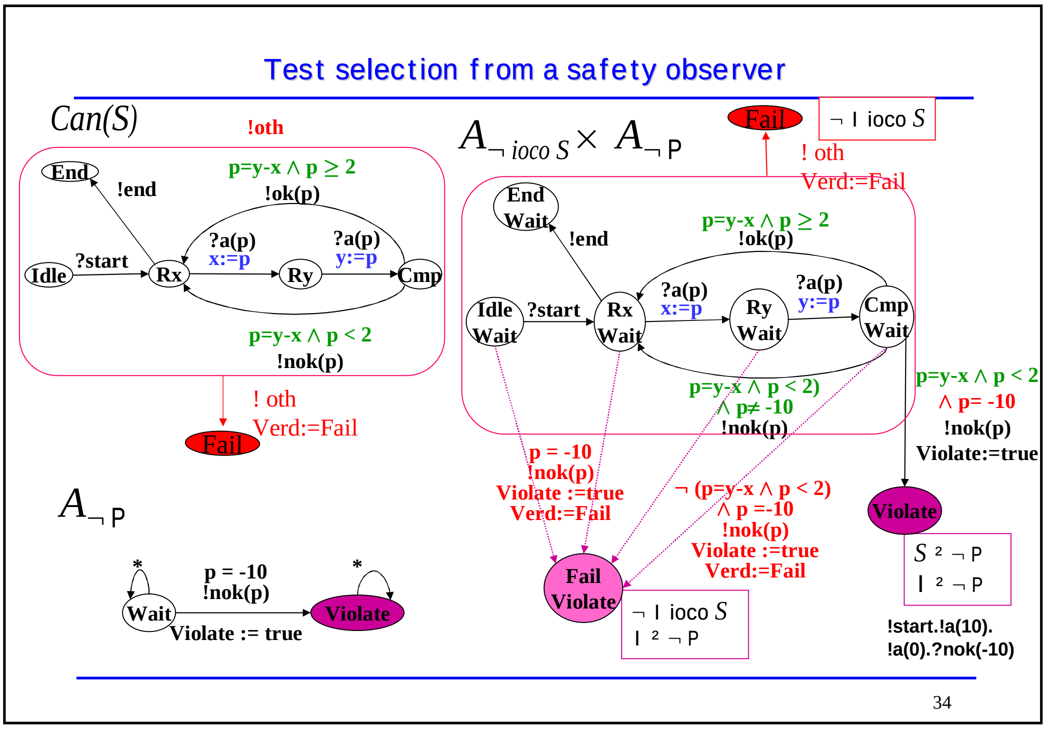
Verification and Testing



32



Model-checking a safety property
S
Model-checking S ² P ? reduces to
reachability in S × A¬ P (undecidable)
End
!end
p=y-x ∧ p ≥ 2
!ok(p)
?a(p)
x:=p
?a(p)
y:=p
Idle
?start
Rx
Ry
Cmp
p=y-x ∧ p < 2
!nok(p)
End Wait
Idle Wait
Rx Wait
Ry Wait
Cmp Wait
p=y-x ∧ p < 2
∧ p = -10
!nok(p)
p=y-x ∧ p < 2
∧ p ≠ -10
!nok(p)
Rx Violate
TP - = A¬ P
p = -10
!nok(p)
Violate := true
Wait
Violate
S ² ¬ P :?start.?a(10).?a(0).!nok(-10) → (Rx, Violate)
With any abstraction Sα ² ¬ P ? is Yes
but Sα ² ¬ P ; S ² ¬ P
Result of S ² P ? could be Unknown
33



Test selection from a safety observer
Can(S)
!oth
End
!end
p=y-x ∧ p ≥ 2
!ok(p)
?a(p)
x:=p
?a(p)
y:=p
Idle
?start
Rx
Ry
Cmp
p=y-x ∧ p < 2
!nok(p)
! oth
Verd:=Fail
Fail
A¬ P
p = -10
!nok(p)
Wait
Violate
Violate := true
A¬ ioco S × A¬ P
Fail
¬ I ioco S
! oth
Verd:=Fail
End Wait
Idle Wait
Rx Wait
Ry Wait
Cmp Wait
p=y-x ∧ p < 2)
∧ p≠ -10
!nok(p)
p=y-x ∧ p < 2
∧ p= -10
!nok(p)
Violate:=true
p = -10
!nok(p)
Violate :=true
Verd:=Fail
¬ (p=y-x ∧ p < 2)
∧ p =-10
!nok(p)
Violate :=true
Verd:=Fail
Fail Violate
¬ I ioco S
I ² ¬ P
Violate
S ² ¬ P
I ² ¬ P
!start.!a(10).
!a(0).?nok(-10)
34

Some links between Model-checking and Conformance Testing

- Test selection using model-checking :
  - S deter., controllable, P reachability: TC ≃ counter-exple of S ² ¬ P

    [Engels et al. 97, Gargantini et al.99]
  - Extension to coverage using CTL [Hong et al.02] or observers [Blom et al.04]
  - Non-controllable case is more complex (this talk)

- Checking properties on the implementation
  - Black-box checking [Peled et al.] : learn I by experiment, model-check I ² P

35

Conclusion

Simplified and general framework for Ioco-based Test selection

- For finite ioLTS and infinite *ioSTS*
- Unified For Reachability and Safety Observers
- Using verification: coreachability analysis, over-approximations
- Completing verification (case of safety)

More research work needed for, e.g.

- Theories and algorithms for other models of reactive systems e.g. with time, data, stack, probabilities …and combinations
- Coverage : measures, selection
- Links with structural testing techniques
- ….

36